Towards the Humanoid Robot Butler

Caitlin Lagrand
Universiteit van Amsterdam
Email: caitlin.lagrand@student.uva.nl

Michiel van der Meer
Universiteit van Amsterdam
Email: michiel.vandermeer@student.uva.nl

***Abstract*—This project focuses on detecting a tomato in a kitchen environment and grabbing it from the table. The ROS framework is used for the implementation of this project. Three detection algorithms are discussed to best detect the tomato. Also a localization algorithm is defined to determine the location of the tomato. The ROS library MoveIt is used for its inverse kinematics to grab the tomato from the table.**

1. Introduction

The use of robots has been increasing over the last years. Not only in the industry, but also in social sectors, such as health care or education. This project focuses on finding ways to apply the help of robots in a domestic environment. It is inspired by the HUMABOT Challenge of 2014 [1]. In this challenge, the kitchen is used as the environment where a robot has to perform the following tasks:

- The safety task: one of the burners in the kitchen is lit and the robot has to turn it off.
- The shopping list: identify missing objects on the shelves to make a shopping list.
- The roasted tomato: identify a tomato and put it into a pan.

This project focuses on "The roasted tomato" task. Section 2 will discuss previous research. In section 3, the used methods will be explained. Section 4 will demonstrate the different simulations that were used. In section 5 the results will be shown. Section 6 will evaluate the project and discuss ideas for future research. Section 7 will sum up the project and conclude.

2. Previous research

This section discusses previous research. Two previously done researches were used: the Cognitive Image Processing (CIP) and some research from previous competitors of the HUMABOT Challenge.

2.1. Cognitive Image Processing (CIP)

Ras [2] used a series of filters and detectors to maximize the accuracy when finding spherical objects. Since the detection is color invariant, the program would be able to find objects more dynamically. In short, it tries to find edges in a blurred image from the saturation channel of the original image. After detecting these edges, the user ends up with a binary image, with the outlines of objects highlighted if the recognition worked correctly. These binary images are then searched for any round shapes using blob detectors. Section 3.1.3 explains how this works. Eventually, a heavily modified version of the CIP-module was considered as one of the three main detectors.

2.2. Previous competitors

Since the HUMABOT Challenge was held in 2014, the teams have no incentive to keep their code private any longer. One of the teams, NAO-UPC, made their implementation publicly available on Github [1]. Even though the team did not complete the challenge, the team's code provided some insights into OpenCV. Their low scores were most likely caused by a very conservative approach to localization and grasping objects. The localization required markers, which indicate the different pieces of furniture in the environment. This means that if the markers were either missing or misplaced, the robot would stop functioning properly.

3. Method

The robot used in this research is the NAO from Aldebaran[2]. The NAO is a 50 cm tall humanoid robot. The environment is the standard play kitchen from IKEA including all the tools and vegetables (see Figure 1).

The ROS framework [3] was used to give access to the many tools and libraries it has and, most importantly, to the MoveIt library [4]. MoveIt introduces a number of tools that aim to create developer-friendly software for mobile manipulation. This library, among other things, incorporates kinematics, motion planning and execution. To process images, the OpenCV library [5] is used. All of these libraries are available in both Python and C++. The code presented with this paper is written in Python. All of our code is available at our Github repository [3].

1. https://github.com/gerardcanal/NAO-UPC
2. https://www.aldebaran.com/en/cool-robots/nao
3. https://github.com/Caiit/tomato_tracker_py

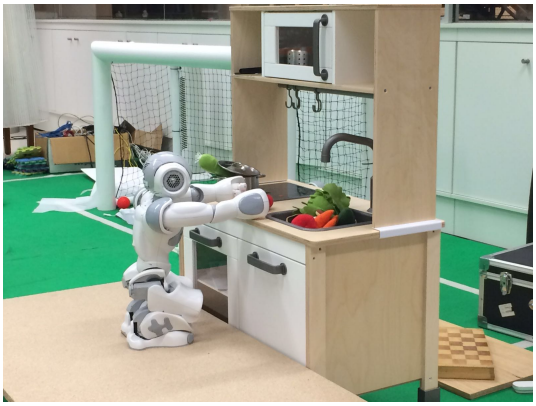


Figure 1. The kitchen environment

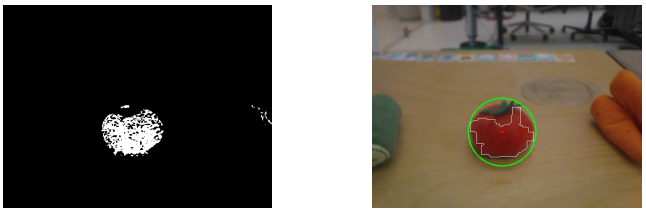
The report is divided into three parts: detecting, localizing and grabbing the tomato. This section explains the methods used for the different parts and the integration of the three parts. In section 3.1, different approaches to detect the tomato will be explained. Section 3.2 focuses on localizing the tomato and section 3.3 will explain the MoveIt implementation to grab the tomato. Section 3.4 will describe the implementation in ROS.

3.1. Detecting

To detect the tomato, three different approaches were used. Two approaches are based on different characteristics of the tomato. The last approach is based on color invariant object recognition.

3.1.1. Color based and finding contours. The first approach focuses on the color of the tomato. Because the tomato is red, the image can be adjusted to filter out everything that is not red (see Figure 2a). This is done by transforming the image into the HSV color space and using the threshold $(0, 140, 60) - (3, 250, 250)$ to get the binary image of red. Keep in mind that this range depends on lighting as well as the camera itself. After removing the noise by using OpenCVs erode and dilate function, contours were found with `findContours()` from OpenCV. From those contours, the minimal enclosing circles were found, because a tomato is more or less a circle (see Figure 2b). From these circles, the "pixel-coordinates" of the center of the tomato and the radius were obtained. The tomato is detected when the radius is bigger than 20 pixels to filter out small red objects in the background. This limitation was put in place, because the surroundings of the robot are not filtered in any way. This means that its camera will pick up anomalies which are not related to the task at hand.

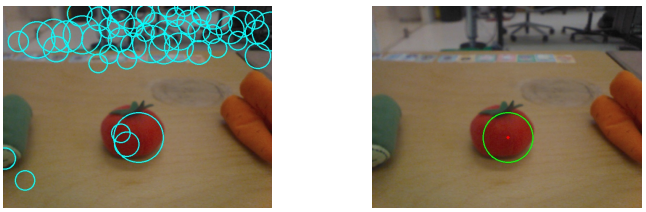
3.1.2. Circle based and average color. The second approach is based on the fact that a tomato is almost round. After transforming the image into the HSV color space and detecting edges with the Canny Edge Detector, circles were detected in the image using HoughCircles (see



(a) Binary image of red pixels

(b) Finding the tomato with findingContous

Figure 2. Color based detection in combination with finding contours



(a) Detecting circles using HoughCircles

(b) The best circle with red as average color

Figure 3. Circle based detection in combination with average color

Figure 3a). The next step is to calculate the average color of the found circles. If the average color is in the range $(20, 20, 70) - (55, 70, 200)$, the circle is considered as red and thus as the tomato (see Figure 3b). The range for red used in this method differs from the one used in the color based approach. This is because after a circle is detected, some small areas inside this circle might not be red, but for instance part of the crown. Because the circle detector has to find a perfect circle and the tomato is not, the color of these areas is weighted into the average color. This makes it not the same shade of red as the tomato, but a slighty adjusted red.

3.1.3. Color invariant and blob detection. The last approach is color invariant. It is a simplified version of CIP. It uses blob detection to detect blobs in the image. In this process, the raw image is parsed to OpenCV's `SimpleBlobDetector()`[4], which will then perform blob detection on the image. These blobs are defined as regions in the image that differ from surrounding regions. The algorithm performs four steps:

1) Thresholding: Converts the image to binary images (images containing only two distinct colors, i.e. black and white) using parameterized thresholds. The image is processed into these binary images, each image containing colors that are inside the thresholds as white, and the colors outside the thresholds as black. This is repeated multiple times, until the entire image is divided.
2) Grouping: For each binary image, the pixels of one color are grouped together. This color is kept the

4. http://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_feature_detectors.html

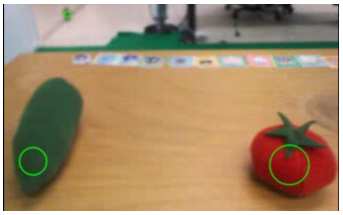


Figure 4. Detecting the tomato using the blobDetector

same across multiple binary images. The centers of these blobs are calculated.

3) Merging: If blobs show up across multiple binary images and their centers are close together, they are considered the same blob and are merged together.
4) Center and Radius calculation: Per blob group the center and radius are then returned to the caller, which can span multiple binary images.

When these centers and radii are returned, it is possible to put constraints on what the blob looks like. For the tomato detector, a largely circular blob would be accepted and a square blob would be rejected.

3.2. Localizing

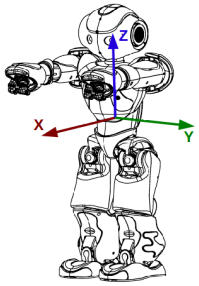
To localize, a simplified method was used instead of using reference worlds. Since it is known that the target object is at a fixed height in the area, a simpler method was both easier to implement and faster to work with. The robots starts out receiving images from his top camera. As long as there is no tomato detected, the robot will turn to the right. This simplified approach might cause issues in a more dynamic environment, where the table is further away from the robot's starting position (1+ meters). At that point, even when the target object comes into vision, either the resolution of the camera would make the object too small to detect or the detector would not consider the generated output as a valid target.

If it does manage to detect the tomato in an image, the x,y-position and the radius of the tomato are found. Now this position needs to be converted to real-world coordinates with the robot as origin (see Figure 5). The z-coordinate is fixed, because the height of the table is known: 0.35m. This is 0.9m in the robot orientation.

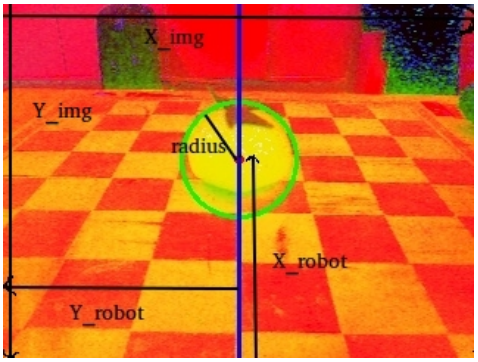
3.2.1. Finding the x-coordinate in the robot orientation. The x-coordinate of the robot is the distance between the robot and the tomato. This is calculated by determining what the radius of the tomato is in an image with the tomato lying on a distance of 0.3 meter. The radius was 36 pixels. The radius for a distance of 1 meter can be derived from those values as shown in (1). This constant can now be used to calculate the x-coordinate (2).

$$RADIUS\_TO\_METERS = 36/0.3 \quad (1)$$

$$distanceX = RADIUS\_TO\_METERS/radius \quad (2)$$



(a) The robots coordinates system



(b) The different orientations shown in one image

Figure 5. The different orientations from the robot and the image

3.2.2. Finding the y-coordinate in the robot orientation. When drawing a vertical line in the middle of the image, the y-coordinate is the offset from this line to the center of the tomato. There are two possibilities: if the tomato is left of this line, the y-coordinate is positive; if it is on the right side, the y-coordinate is negative.

To get this offset in pixels, the x-position of the image is subtracted from this line (3).

Now those pixels must be converted to meters. This is done using the real width of the tomato, which is 0.05m (4).

Multiplying this with the offset in pixels will result in the y-coordinate of the robot (5).

$$offsetInPixels = (IMG\_WIDTH/2) - x \quad (3)$$

$$pixelToMeter = REAL\_WIDTH/(radius * 2) \quad (4)$$

$$offsetY = pixelToMeter * offsetInPixels \quad (5)$$

3.3. Grabbing

Grabbing the tomato is done by MoveIt. As mentioned before, this library contains a number of tools for mobile manipulation. Only the left and right arm groups are used. Given the center point of the tomato, MoveIt first moves the arms of the robot 5cm to the left or to the right of the tomato, but it does not grab it yet. This is done to correct any possible localization errors. Next, both arms are moved towards each other with 2cm to squeeze the tomato to grab

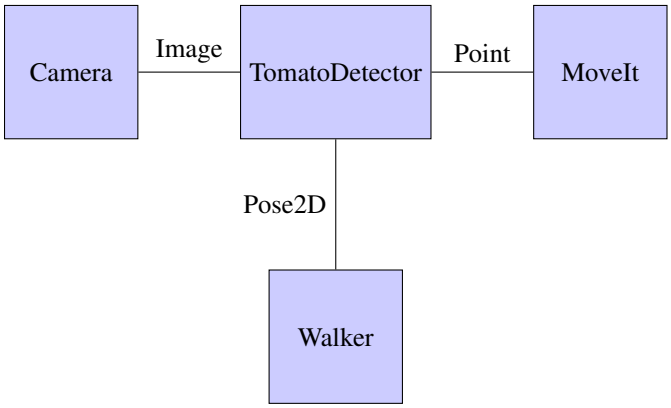


Figure 6. The implementation in ROS showing the used nodes and topics

it. Finally, the arms are lifted up a bit and moved towards the chest to avoid hitting the table and to be more stable while moving.

3.4. Implementation in ROS

The ROS-implementation consists of four nodes (see Figure 6). The main node is the TomatoDetector node. This node is a subscriber to the Image topic and a publisher to the Point and Pose2D topics. The Camera and Walker nodes start while running the nao_bringup for the connection with the NAO. Those nodes are used for the images and to walk. The MoveIt node is used for inverse kinematics to grab the tomato.
Our ROS implementation for the NAO depends on the following packages:

- nao_robot
- vision_opencv
- naoqi_bridge
- nao_moveit_config
- nao_dcm_robot
- nao_virtual

3.4.1. The idea behind the implementation. The Camera node publishes images taken by the NAO. The TomatoDetector is subscribed to those images, so it receives them. Firstly, it tries to find the tomato in the image. If it does not find the tomato, it turns right and searches again. If it does find the tomato, it checks whether it is close enough to grab it. If so, it publishes a Point and the MoveIt node will start grabbing the tomato. However when it is not close enough to grab it directly, it publishes a Pose2D to move closer to the tomato. This Pose2D has the obtained x-coordinate of the tomato minus 0.3m as its x-coordinate, otherwise the robot will hit the table. The y-coordinate is the y-coordinate of the tomato. When moving closer to the tomato, it will search for the tomato again to check if it is now close enough to grab the tomato. This is done to compensate for possible errors during localization.

4. Simulations

Most of the software this paper discusses can be used within simulations. For the implementation described in this paper, two different simulators were used primarily:

- Webots
- RVIZ

4.1. Webots

Webots is a commercial robot simulator promoted by the HUMABOT organizers. They made the working environment available in a format that can be used in Webots, such that developers that do not have access to physical robots can still practice programming one. However, it became apparent that in order to work with Webots, new controllers for the robots would have to be written. After considering this, it was decided that this would take too much time. Therefore, Webots was dropped as a simulator.

4.2. RVIZ

RVIZ is not an actual simulator, but a 3D visualization tool integrated with ROS. The program loads configurations that define what kind of objects are in the world and gets the data from ROS nodes. By simply listening to the broadcast data it is then able to show in what way everything is moving. By sending back data, for instance two objects colliding, it sends information to the relevant nodes.

Due to the architecture of different nodes, controllers are not necessary. Instead, it is possible to pretend that a real robot is sending information to RVIZ. This way, while RVIZ thinks a real robot is connected, it is just receiving broadcast data from a robot that is running virtually. By design, it is then also possible to actually plug in a physical robot, and have the program execute without significant changes, aside from recalibration.

One drawback from RVIZ was that the environment was not readily available, and had to be converted manually. This meant exporting all the objects to a format that was readable by RVIZ with Blender. Blender is 3D graphics software that supports multiple plugins, one of which is an automated converter. [5] After converting, all object lost their original position and had to be manually readjusted.

With RVIZ working and the robot connected, it is now possible to simulate the robot grabbing a tomato at a specified point, since MoveIt is also controlled using the ROS system (see Figure 4.2. The specification of this point would be done by the detector program.

5. Results

The results described in this section are split into three parts. This is because these three parts form a core to our project. If one part was unreliable, the other would suffer

5. https://www.blender.org/

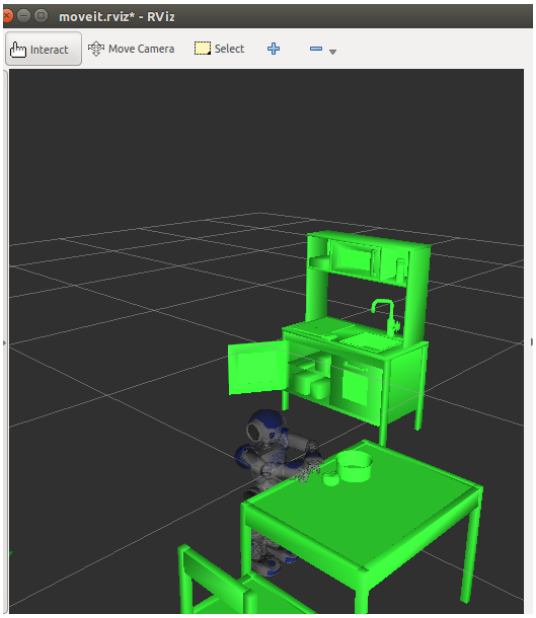


Figure 7. A screenshot of the RVIZ visualization

from it. Firstly, the tomato detectors are evaluated in section 5.1. Then, with the best performing detector, the localization algorithm is evaluated in section 5.2 and section 5.3 will go into detail about the results of grabbing the tomato. This is not subjected to much theoretical testing, as it has to perform well in a physical setting. This means that we did not give the task a measure on how well it was performed.

5.1. Tomato detector evaluation

The three detection algorithms are tested on ten different images: five images containing only one vegetable (tomato, carrot, cucumber, garlic or lettuce), three images containing all vegetables and two images containing all vegetables without the tomato. Table 1 shows the results of the different detection algorithms. "1" means that it detected the tomato and "0" means that it did not detect anything. If something else was detected, it will say what this was. As can be seen in this table, the color based method always detects the tomato and does not detect anything else. The circle based approach does detect the tomato most of the time but it detects something in the background as a tomato once. The blobs based method detects the tomato most of the time, but detects a lot of other vegetables as the tomato as well. The color based method is the best method and is used for the rest of project.

5.2. Tomato localization evaluation

In order to create a sensible evaluation, a chessboard was used to make sure the tomato was put in the same place every time. The squares on this chessboard are each of a fixed length, making it very easy to create a discrete space on the surface. Table 2 shows the difference between the

TABLE 1. RESULTS OF THE DIFFERENT DETECTING ALGORITHMS
1 MEANS DETECTED TOMATO, 0 MEANS DETECTED NOTHING

	color based	Circle based	Blobs
tomato	1	1	1
carrot	0	0	0
cucumber	0	0	0
garlic	0	background	0
lettuce	0	0	lettuce
all1	1	1	1
all2	1	0	garlic
all3	1	1	1 & background
without1	0	0	garlic
without2	0	0	carrot

detected point and the actual point of the tomato. As can be seen in this table it does not work perfectly. However when looking at the differences, these are often not very large. Since there are ways to compensate those errors, they are disregarded.

5.3. Tomato grabber evaluation

Getting MoveIt to run on a physical NAO proved more difficult than anticipated. Even with the ROS interface being able to easily swap between a simulation and a real NAO, the actuators that are controlled by MoveIt did not correspond to any on the physical NAO. This resulted in being unable to run a complete test including the tomato grabbing.

However, the simulation did show great results. The animation was able to grab the tomato after specifying the location the tomato detector thought it was at.

6. Discussion

After testing three different detectors, the one that performed best was picked, which was the color based and finding contours detector. Since both the detector and the localization were interfaced with ROS, it was possible to create a rosbag. This rosbag contains information about the location the program thought the tomato was during tests. It is possible to play back the stored temporal information by reading the file[6].

Even thought the localization seemed to work within a certain error margin, future research might want to implement a more sophisticated process. The simple approach worked because we optimized the algorithm to our environment. When looking at the future, a new method is suggested where the x distance from the robot is not calculated using the somewhat unreliable size, but for instance a position derived from multiple viewpoints.

6. http://wiki.ros.org/rosbag

TABLE 2. RESULTS OF THE LOCALIZATION ALGORITHMS, IN METERS

Real x	Real y	Estimated x	Estimated y	Difference x	Difference y
0.224	0	0.225	-0.00572916666667	0.001	-0.00572916666667
0.224	0.076	0.27	0.076875	0.046	0.000875
0.224	-0.076	0.27	-0.07625	0.046	0.00025
0.30	0	0.308571428571	0.0	0.008571429	0.0
0.30	0.076	0.308571428571	0.0617857142857	0.008571429	0.014214286
0.30	-0.076	0.385714285714	-0.0709077426365	0,085714286	0,005092257
0.376	0	0.372413793103	0.000862068965517	0.003586207	0.000862068965517
0.376	0.076	0.372413793103	0.0614359099289	0.003586207	0.01456409
0.376	-0.076	0.54	-0.07625	0.164	0.00025

7. Conclusion

Multiple object recognition methods were explored in this project and three were implemented and tested. The color invariant detector does not work nearly as good as the color based detectors and thus was no longer considered. Using the best of the two left over detectors, the location of the tomato was estimated using pixel values. The location was fed to an inverse kinematics solver, which would then be able to grab the tomato and await further instructions. We view this as the first steps to a dynamic robot cook.

Acknowledgments

The authors would like to thank Arnoud Vissser.

References

[1] P. J. S. Enric Cervera, Juan Carlos Garcia, "Toward the robot butler: The humabot challenge," Robotics & Automation Magazine, IEEE (Volume:22, Issue: 2), 2015.

[2] G. Ras, "Cognitive image processing for humanoid soccer in dynamic environments," 2015.

[3] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[4] S. C. Sachin Chitta, Ioan Sucan, "Moveit!" Robotics & Automation Magazine, IEEE (Volume:20, Issue: 1), 2012.

[5] G. Bradski, *Dr. Dobb's Journal of Software Tools*.